UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/729.767	12/08/2003	Richard P. Himmer	120 04983US	3406

128 7590 03/08/2007		EXAMINER
HONEYWELL INTERNATIONAL INC.		WANG, BEN C
101 COLUMBIA ROAD		
P O BOX 2245		
MORRISTOWN, NJ 07962-2245		

ART UNIT	PAPER NUMBER
2192	

SHORTENED STATUTORY PERIOD OF RESPONSE	MAIL DATE	DELIVERY MODE
3 MONTHS	03/08/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

If NO period for reply is specified above, the maximum statutory period will apply and will expire 6 MONTHS from the mailing date of this communication.

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE <u>3</u> MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.
- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on <u>08 December 2003</u>.

2a)☐ This action is **FINAL**.   2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) <u>1-17</u> is/are pending in the application.

    4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) <u>1-17</u> is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

    Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

    Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

    a)☐ All   b)☐ Some * c)☐ None of:

      1.☐ Certified copies of the priority documents have been received.

      2.☐ Certified copies of the priority documents have been received in Application No. _____.

      3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

    * See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1)☒ Notice of References Cited (PTO-892)

2)☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3)☐ Information Disclosure Statement(s) (PTO/SB/08)
    Paper No(s)/Mail Date _____.

4)☐ Interview Summary (PTO-413)
    Paper No(s)/Mail Date. _____ .

5)☐ Notice of Informal Patent Application

6)☐ Other: _____.

DETAILED ACTION

1.   Claims 1-17 are pending in this application and presented for examination.

Claim Rejections – 35 USC § 101

2.   35 U.S.C. 101 reads as follows:

> Whoever invents or discovers any new and useful process, machine, manufacture, or
> composition of matter, or any new and useful improvement thereof, may obtain a patent therefor,
> subject to the conditions and requirements of this title.

3.   Claims 11-15 are rejected under 35 U.S.C 101 because the claims are directed

to non-statutory subject matter.

4.   **As to claim 11**, "a data structure" is being cited, line 1; however, it appears that it

would reasonably be interpreted by one of ordinary skill in the art as non-functional

descriptive material per se. Mere arrangements or compilations of facts or data, without

any functional interrelationship is not a process, machine, manufacture, or composition

of matter. Nonfunctional descriptive material that does not constitute a statutory

process, machine, manufacture, or composition of matter, and should be rejected under

35 U.S.C. 101. (See MPEP 2106.01(II)).

5.   **As to claims 12-15**, they are merely further recited as the feature of the data

structure per se, thus, do not cure the deficiency of claim 11, and also rejected under 35

U.S.C. 101 as set forth above.

Claim Rejections – 35 USC § 102(b)

6.   The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102(b)

that form the basis for the rejections under this section made in this office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

7.   Claim 16 is rejected under 35 U.S.C. 102(b) as being anticipated by Almond et

al. (Patent No. 6,112,024) (hereinafter 'Almond')

8.   **As to claim 16**, Almond discloses a computer readable medium having

executable instructions stored thereon to perform a method of determining object

relationships when checking-in, the method comprising determining whether an object

to be checked-in has a first derivation parent (Col. 3, Lines 39-53 – if an object instance

links to multiple other objects (e.g., a .c file which links to three .h files), the object

instance would maintain a separate link for each dependent object, for representing that

particular relationship); adding a name (Fig. 9, element 915 – *NodeName*; Col. 9, Lines

62-63 – each node contains a name, node name) and a version (Fig. 9, element of 945

– *VersionString*) of the first derivation parent to a list of object relationships (Col. 9,

Lines 57-59 – each entry in the table is represented by a node ID, for uniquely

identifying the node; in addition, each node includes an ID for its parent – that is parent

node ID; Col. 3, Lines 39-53 – if an object instance links to multiple other objects (e.g., a

.c file which links to three .h files), the object instance would maintain a separate link for

each dependent object, for representing that particular relationship), if the object has the

first derivation parent; determining for each contained object that is contained in the

object, whether the contained object has a second derivation parent, if the object does

not have the first derivation parent (Col. 3, Lines 39-53 – if an object instance links to

multiple other objects (e.g., a .c file which links to three .h files), the object instance

would maintain a separate link for each dependent object, for representing that

particular relationship); and adding a name (Fig. 9, element 915 – *NodeName*; Col. 9,

Lines 62-63 – each node contains a name, node name) and a version of the second

derivation parent to the list of object relationships (Fig. 9, element of 945 –

VersionString), if the contained object has the second derivation parent; providing the

list of object relationships (Col. 9, Lines 57-59 – each entry in the table is represented

by a node ID, for uniquely identifying the node; in addition, each node includes an ID for

its parent – that is parent node ID; Col. 3, Lines 39-53 – if an object instance links to

multiple other objects (e.g., a .c file which links to three .h files), the object instance

would maintain a separate link for each dependent object, for representing that

particular relationship).

Claim Rejections – 35 USC § 103(a)

9.   The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all

obviousness rejections set forth in this office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in
> section 102 of this title, if the differences between the subject matter sought to be patented and the prior art
> are such that the subject matter as a whole would have been obvious at the time the invention was made to
> a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be
> negatived by the manner in which the invention was made

10.   Claims 1-15, and 17 are rejected under 35 U.S.C. 103(a) as being unpatentable

over Almond in view of Watkins et al. (Pub No. US 2001/0054042 A1) (hereinafter

'Watkins')

11.   **As to claim 1**, Almond discloses a source control system (Fig. 1B; Col. 5, Lines

9-16 – an "object Cycle" versioning system; Fig. 2) for a process control system,

comprising: a processor in a process control system (Fig. 1A, element 101 – Central

Processor); a database accessible (Col. 5, Lines 32-36 – through a database interface

layer (ODBC); Fig. 2, element of 220 – RDBMS) by the processor to store information

(Abstract, Lines 9-15) associated with an object under source control to be checked-out

(Abstract, Lines 18-21– versioning activities can be undertaken, such as checking out;

Col. 1, Lines 36-41); a check-out function operable on the processor to check-out the

object (Abstract, Lines 18-21 – versioning activities can be undertaken, such as

checking out; Col. 1, Lines 36-41), to use the information to determine whether any

dependent objects exist (Col. 3, Lines 39-53 – if an object instance links to multiple

other objects (e.g., a .c file which links to three .h files), the object instance would

maintain a separate link for each dependent object, for representing that particular

relationship).

Although Almond discloses using the information to determine whether any

dependent objects exist (Col. 3, Lines 39-53 – if an object instance links to multiple

other objects (e.g., a .c file which links to three .h files), the object instance would

maintain a separate link for each dependent object, for representing that particular

relationship), but Almond does not explicitly disclose automatically checking-out the

existing dependent objects.

However, in an analogous art of computing system for information management,

Watkins discloses automatically checking-out the existing dependent objects ([0071] –

when the parent folder is checked out, all of its children and descendants are also

required to be checked out; [0072] – this recursion is preferably implemented by making

the object aware of not only its metadata, but of its parent-child relationship with other

objects in the system; Figs. 10-11; [0102]-[0104]).

Therefore, it would have been obvious to one of ordinary skill in the art, at the

time the invention was made to combine the teachings of Almond and the teachings of

Watkins to further provide automatically checking-out the existing dependent objects in

Almond system.

The motivation is that it would provide and/or enhance the Almond system as

"rather than present a sea of information… to the user, the user only see what is

currently being worked on (i.e., checkout out), which in this case is a workbin… and to

ensure data integrity", as once suggested by Watkins (i.e., [0085]-[0086]).

12. **As to claim 5**, Almond discloses a method of automatic check-out (Abstract,

Lines 18-21– versioning activities can be undertaken, such as checking out; Col. 1,

Lines 36-41) for a source control system (Fig. 1B; Col. 5, Lines 9-16 – an "object Cycle"

versioning system; Fig. 2) in a process control system, comprising: storing information

associated with an object (Abstract, Lines 9-15); receiving a request from a user to

check-out the object (Fig. 8, elements of 810 – Communication Layer, 825 – Change

Control, 820 – Session Management/Verification; Col. 8, Lines 51-54, 58-61);

determining whether any dependent objects of the object exist based on the information

(Col. 3, Lines 39-53 – if an object instance links to multiple other objects (e.g., a .c file

which links to three .h files), the object instance would maintain a separate link for each

dependent object, for representing that particular relationship) and providing a status to

the user (Fig. 1B, element of 160 – User Interface and USER).

Although Almond discloses using the information to determine whether any

dependent objects exist (Col. 3, Lines 39-53 – if an object instance links to multiple

other objects (e.g., a .c file which links to three .h files), the object instance would

maintain a separate link for each dependent object, for representing that particular

relationship), but Almond does not explicitly disclose automatically checking-out the

existing dependent objects when the object is checked-out.

However, in an analogous art of computing system for information management,

Watkins discloses automatically checking-out the existing dependent objects when the

object is checked-out ([0071] – when the parent folder is checked out, all of its children

and descendants are also required to be checked out; [0072] – this recursion is

preferably implemented by making the object aware of not only its metadata, but of its

parent-child relationship with other objects in the system; Figs. 10-11; [0102]-[0104]).

Therefore, it would have been obvious to one of ordinary skill in the art, at the

time the invention was made to combine the teachings of Almond and the teachings of

Watkins to further provide automatically checking-out the existing dependent objects

when the object is checked-out in Almond system.

The motivation is that it would provide and/or enhance the Almond system as "rather than present a sea of information... to the user, the user only see what is currently being worked on (i.e., checkout out), which in this case is a workbin... and to ensure data integrity", as once suggested by Watkins (i.e., [0085]-[0086]).

13.   **As to claim 11**, Almond discloses a data structure (Fig. 9 – a block diagram illustrating a relational schema employed for internal storage and management of versioned objects in the system) for automatic check-out, comprising: a reference object name to identify the object to be checked-out (Fig. 9, element 915 – *NodeName*; Col. 9, Lines 62-63 – each node contains a name, node name for representing an entity); a reference object type of the object (Fig. 9, element 932 – *Entity Type*; Col. 10, Lines 12-14 – the entities table also stores a type, entity type, for characterizing each entity as user data, a label, a branch label, or a group); and a reference type of the object (Fig. 9, element 913 – *ParentNodeID*; Col. 9, Lines 58-59 – each node includes an ID for its parent – that is , parent node ID).

Although Almond discloses using the information to determine whether any dependent objects exist (Col. 3, Lines 39-53 – if an object instance links to multiple other objects (e.g., a .c file which links to three .h files), the object instance would maintain a separate link for each dependent object, for representing that particular relationship), but Almond does not explicitly disclose an automatic check-out function automatically checks-out dependent objects based on the reference type.

However, in an analogous art of computing system for information management, Watkins discloses an automatic check-out function automatically checks-out dependent objects based on the reference type.

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Almond and the teachings of Watkins to further provide an automatic check-out function automatically checks-out dependent objects based on the reference type in Almond system.

The motivation is that it would provide and/or enhance the Almond system as "rather than present a sea of information… to the user, the user only see what is currently being worked on (i.e., checkout out), which in this case is a workbin… and to ensure data integrity", as once suggested by Watkins (i.e., [0085]-[0086]).

14.   **As to claim 17**, Almond discloses a computer readable medium having executable instructions stored thereon to perform a method of automatic check-out for a source control system in a process control system, the method comprising storing information associated with an object (Abstract, Lines 9-15); receiving a request from a user to check-out the object (Fig. 8, elements of 810 – Communication Layer, 825 – Change Control, 820 – Session Management/Verification; Col. 8, Lines 51-54, 58-61); determining whether any dependent objects of the object exist based on the information (Col. 3, Lines 39-53 – if an object instance links to multiple other objects (e.g., a .c file which links to three .h files), the object instance would maintain a separate link for each dependent object, for representing that particular relationship); and providing a status to the user (Fig. 1B, element of 160 – User Interface and USER).

Although Almond discloses using the information to determine whether any dependent objects exist (Col. 3, Lines 39-53 – if an object instance links to multiple other objects (e.g., a .c file which links to three .h files), the object instance would maintain a separate link for each dependent object, for representing that particular relationship), but Almond does not explicitly disclose automatically checking-out the existing dependent objects when the object is checked-out.

However, in an analogous art of computing system for information management, Watkins discloses automatically checking-out the existing dependent objects when the object is checked-out ([0071] – when the parent folder is checked out, all of its children and descendants are also required to be checked out; [0072] – this recursion is preferably implemented by making the object aware of not only its metadata, but of its parent-child relationship with other objects in the system; Figs. 10-11; [0102]-[0104]).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Almond and the teachings of Watkins to further provide automatically checking-out the existing dependent objects when the object is checked-out in Almond system.

The motivation is that it would provide and/or enhance the Almond system as "rather than present a sea of information... to the user, the user only see what is currently being worked on (i.e., checkout out), which in this case is a workbin... and to ensure data integrity", as once suggested by Watkins (i.e., [0085]-[0086]).

15.   **As to claim 2**, although Almond discloses using the information to determine

whether any dependent objects exist (Col. 3, Lines 39-53 – if an object instance links to

multiple other objects (e.g., a .c file which links to three .h files), the object instance

would maintain a separate link for each dependent object, for representing that

particular relationship), but Almond does not explicitly disclose the system according

further comprising: a propagation function operable on the processor to propagate

changes made to the object, the existing dependent objects, when the object is saved.

However, in an analogous art of computing system for information management,

Watkins discloses the system according further comprising: a propagation function

operable on the processor to propagate changes made to the object, the existing

dependent objects, when the object is saved ([0094]; [0071] – when the parent folder is

checked out, all of its children and descendants are also required to be checked out;

[0072] – this recursion is preferably implemented by making the object aware of not only

its metadata, but of its parent-child relationship with other objects in the system; Figs.

10-11; [0102]-[0104]).

Therefore, it would have been obvious to one of ordinary skill in the art, at the

time the invention was made to combine the teachings of Almond and the teachings of

Watkins to further provide the system according further comprising: a propagation

function operable on the processor to propagate changes made to the object, the

existing dependent objects, when the object is saved in Almond system.

The motivation is that it would provide and/or enhance the Almond system as

"rather than present a sea of information... to the user, the user only see what is

currently being worked on (i.e., checkout out), which in this case is a workbin... and to

ensure data integrity", as once suggested by Watkins (i.e., [0085]-[0086]).

16.   **As to claim 3**, Almond discloses the system according wherein the stored

information includes a reference to a parent object (Fig. 9, element 913 –

ParentNodeID; Col. 9, Lines 58-59 – each node includes an ID for its parent – that is ,

parent node ID).

17.   **As to claim 4**, Almond discloses the system according wherein the stored

information is at least one selected from the group consisting of: a name (Fig. 9,

element 915 – *NodeName*; Col. 9, Lines 62-63 – each node contains a name, node

name), a version number (Fig. 9, element of 945 – *VersionString*), a type (Fig. 9,

element 932 – *Entity Type*; Col. 10, Lines 12-14 – the entities table also stores a type,

entity type, for characterizing each entity as user data, a label, a branch label, or a

group) and a status (Fig. 9, 925 – *Status*).

18.   **As to claim 6**, Almond discloses the method further comprising: sorting the

existing dependent objects so that parents precede children ([0071] – when the parent

folder is checked out, all of its children and descendants are also required to be

checked out; [0072] – this recursion is preferably implemented by making the object

aware of not only its metadata, but of its parent-child relationship with other objects in

the system; Figs. 10-11; [0102]-[0104]).

19.   **As to claim 7**, although Almond discloses using the information to determine

whether any dependent objects exist (Col. 3, Lines 39-53 – if an object instance links to

multiple other objects (e.g., a .c file which links to three .h files), the object instance

would maintain a separate link for each dependent object, for representing that

particular relationship), but Almond does not explicitly disclose the method wherein one

of the existing dependent objects is a derivation child of the object.

However, in an analogous art of computing system for information management,

Watkins discloses the method wherein one of the existing dependent objects is a

derivation child of the object ([0071] – when the parent folder is checked out, all of its

children and descendants are also required to be checked out; [0072] – this recursion is

preferably implemented by making the object aware of not only its metadata, but of its

parent-child relationship with other objects in the system; Figs. 10-11; [0102]-[0104]).

Therefore, it would have been obvious to one of ordinary skill in the art, at the

time the invention was made to combine the teachings of Almond and the teachings of

Watkins to further provide the method wherein one of the existing dependent objects is

a derivation child of the object in Almond system.

The motivation is that it would provide and/or enhance the Almond system as

"rather than present a sea of information... to the user, the user only see what is

currently being worked on (i.e., checkout out), which in this case is a workbin... and to

ensure data integrity", as once suggested by Watkins (i.e., [0085]-[0086]).

20.   **As to claim 8**, although Almond discloses using the information to determine

whether any dependent objects exist (Col. 3, Lines 39-53 – if an object instance links to

multiple other objects (e.g., a .c file which links to three .h files), the object instance

would maintain a separate link for each dependent object, for representing that

particular relationship), but Almond does not explicitly disclose the method further

comprising: automatically checking-out a derivation child only if a derivation child is

checked-in.

However, in an analogous art of computing system for information management,

Watkins discloses the method further comprising: automatically checking-out a

derivation child only if a derivation child is checked-in ([0071] – when the parent folder is

checked out, all of its children and descendants are also required to be checked out;

[0072] – this recursion is preferably implemented by making the object aware of not only

its metadata, but of its parent-child relationship with other objects in the system; Figs.

10-11; [0102]-[0104]).

Therefore, it would have been obvious to one of ordinary skill in the art, at the

time the invention was made to combine the teachings of Almond and the teachings of

Watkins to further provide the method further comprising: automatically checking-out a

derivation child only if a derivation child is checked-in in Almond system.

The motivation is that it would provide and/or enhance the Almond system as

"rather than present a sea of information… to the user, the user only see what is

currently being worked on (i.e., checkout out), which in this case is a workbin… and to

ensure data integrity", as once suggested by Watkins (i.e., [0085]-[0086]).

21.   **As to claim 9**, although Almond discloses using the information to determine

whether any dependent objects exist (Col. 3, Lines 39-53 – if an object instance links to

multiple other objects (e.g., a .c file which links to three .h files), the object instance

would maintain a separate link for each dependent object, for representing that

particular relationship), but Almond does not explicitly disclose the method further

comprising: automatically checking-out any children of the object, when the object is a

user-defined template.

However, in an analogous art of computing system for information management,

Watkins discloses the method further comprising: automatically checking-out any

children of the object ([0071] – when the parent folder is checked out, all of its children

and descendants are also required to be checked out; [0072] – this recursion is

preferably implemented by making the object aware of not only its metadata, but of its

parent-child relationship with other objects in the system; Figs. 10-11; [0102]-[0104]),

when the object is a user-defined template ([0114]).

Therefore, it would have been obvious to one of ordinary skill in the art, at the

time the invention was made to combine the teachings of Almond and the teachings of

Watkins to further provide the method further comprising: automatically checking-out

any children of the object, when the object is a user-defined template in Almond system.

The motivation is that it would provide and/or enhance the Almond system as

"rather than present a sea of information… to the user, the user only see what is

currently being worked on (i.e., checkout out), which in this case is a workbin... and to

ensure data integrity", as once suggested by Watkins (i.e., [0085]-[0086]).

22.   **As to claim 10**, although Almond discloses using the information to determine

whether any dependent objects exist (Col. 3, Lines 39-53 – if an object instance links to

multiple other objects (e.g., a .c file which links to three .h files), the object instance

would maintain a separate link for each dependent object, for representing that

particular relationship), but Almond does not explicitly disclose the method further

comprising: automatically checking-out any children of the children of the object, when

the children are user-defined templates.

However, in an analogous art of computing system for information management,

Watkins discloses the method further comprising: automatically checking-out any

children of the children of the object ([0071] – when the parent folder is checked out, all

of its children and descendants are also required to be checked out; [0072] – this

recursion is preferably implemented by making the object aware of not only its

metadata, but of its parent-child relationship with other objects in the system; Figs. 10-

11; [0102]-[0104]), when the children are user-defined templates ([0114]).

Therefore, it would have been obvious to one of ordinary skill in the art, at the

time the invention was made to combine the teachings of Almond and the teachings of

Watkins to further provide the method further comprising: automatically checking-out

any children of the children of the object, when the children are user-defined templates

in Almond system.

The motivation is that it would provide and/or enhance the Almond system as "rather than present a sea of information... to the user, the user only see what is currently being worked on (i.e., checkout out), which in this case is a workbin... and to ensure data integrity", as once suggested by Watkins (i.e., [0085]-[0086]).

23.   **As to claim 12**, Almond discloses the data structure further comprising a parent object (Col. 9, Lines 57-59 – each entry in the table is represented by a node ID, for uniquely identifying the node; in addition, each node includes an ID for its parent – that is parent node ID; Col. 3, Lines 39-53 – if an object instance links to multiple other objects (e.g., a .c file which links to three .h files), the object instance would maintain a separate link for each dependent object, for representing that particular relationship).

24.   **As to claim 13**, although Almond discloses using the information to determine whether any dependent objects exist (Col. 3, Lines 39-53 – if an object instance links to multiple other objects (e.g., a .c file which links to three .h files), the object instance would maintain a separate link for each dependent object, for representing that particular relationship), but Almond does not explicitly disclose the data structure further comprising a parent version.

However, in an analogous art of computing system for information management, Watkins discloses the data structure further comprising a parent version (Fig. 10, element 615-5 – *parentVerNum*; Fig. 11, element 625-5 – *parentVerNum*).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Almond and the teachings of

Watkins to further provide the data structure further comprising a parent version in
Almond system.

The motivation is that it would provide and/or enhance the Almond system as
"rather than present a sea of information... to the user, the user only see what is
currently being worked on (i.e., checkout out), which in this case is a workbin... and to
ensure data integrity", as once suggested by Watkins (i.e., [0085]-[0086]).

25.   **As to claim 14,** Almond discloses the data structure wherein the reference
object type is composite or basic (Fig. 9, element 932 – *entity type*; Col. 10, Lines 12-
14).

26.   **As to claim 15,** Almond discloses the data structure wherein the reference type
is parent (Col. 9, Lines 58-59 – each node includes an ID for its parent).

Although Almond discloses using the information to determine whether any
dependent objects exist (Col. 3, Lines 39-53 – if an object instance links to multiple
other objects (e.g., a .c file which links to three .h files), the object instance would
maintain a separate link for each dependent object, for representing that particular
relationship), but Almond does not explicitly disclose the data structure wherein the
reference type is contained child.

However, in an analogous art of computing system for information management,
Watkins discloses the data structure wherein the reference type is contained child (Fig.
10 – element 615-2 – *childObjID*; Fig. 11, element 625-2 - *childObjID*).

Therefore, it would have been obvious to one of ordinary skill in the art, at the

time the invention was made to combine the teachings of Almond and the teachings of

Watkins to further provide the data structure wherein the reference type is contained

child in Almond system.

The motivation is that it would provide and/or enhance the Almond system as

"rather than present a sea of information… to the user, the user only see what is

currently being worked on (i.e., checkout out), which in this case is a workbin… and to

ensure data integrity", as once suggested by Watkins (i.e., [0085]-[0086]).

Conclusion

27.   The prior art made of record and not relied upon is considered pertinent to

applicant's disclosure.

- Schmidt et al., *Software Version Management System* (Pat. No. 4,558,413)

- K. Gao, *Methods and Apparatus for Globalizing Software* (Pat. No. US 6,983,238

   B2)

28.   Any inquiry concerning this communication or earlier communications from the

examiner should be directed to Ben C. Wang whose telephone number is 571-270-

1240. The examiner can normally be reached on Monday - Friday, 8:00 a.m. - 5:00

p.m., EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's

supervisor, Tuan Q. Dam can be reached on 571-272-3695. The fax phone number for

the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the

Patent Application Information Retrieval (PAIR) system. Status information for

published applications may be obtained from either Private PAIR or Public PAIR.

Status information for unpublished applications is available through Private PAIR only.

For more information about the PAIR system, see http://pair-direct.uspto.gov. Should

you have questions on access to the Private PAIR system, contact the Electronic

Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a

USPTO Customer Service Representative or access to the automated information

system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

TUAN DAM
SUPERVISORY PATENT EXAMINER

BCW

February 18, 2007